UVA558

The problem asks whether a scientist can exploit a network of wormholes to travel infinitely into the past. Each wormhole is a one-way connection between star systems and carries a “time shift,” which can be positive (moving forward) or negative (moving backward). The main task is to check if there exists a negative weight cycle reachable from the starting star system (node 0), because such a cycle would let the scientist repeatedly traverse it to go further back in time indefinitely.

We can model the scenario as a directed graph: each star system is a node, and each wormhole is a directed edge weighted by its time shift. Detecting a negative cycle reachable from the start can be efficiently done using the Bellman-Ford algorithm. By initializing distances to all nodes as infinity except the start node (set to 0), we relax all edges for n-1 iterations. Then, in an extra iteration, if any distance can still be reduced, it confirms the presence of a negative cycle reachable from the start node, and we can report that traveling back in time infinitely is “possible.” Otherwise, we report “not possible.”

The approach can be summarized as:

1. **Problem Understanding**
   - Determine if a reachable cycle exists where the sum of time shifts is negative.
   - Such a cycle allows unlimited backward time travel.
2. **Graph Representation**
   - Nodes represent star systems.
   - Directed edges represent wormholes, weighted by their time shifts.
3. **Bellman-Ford Algorithm**
   - Start from node 0 with a distance of 0.
   - Keep track of the earliest arrival time at each node.
   - Relax all edges repeatedly to update minimum times.
4. **Negative Cycle Detection**
   - After n-1 relaxations, do one more iteration.
   - If any distance decreases further, it indicates a negative cycle is reachable from the start.

5. **Result Interpretation**
    - If a negative cycle is detected, output “possible.”
    - Otherwise, output “not possible.”

Input:

4 4

0 1 10

1 2 20

2 3 30

3 0 -60

Execution:

Edges stored as (u, v, weight):

• Edge 0: (0, 1, 10) • Edge 1: (1, 2, 20)

• Edge 2: (2, 3, 30)

• Edge 3: (3, 0, -60)

Step 1: Initialization

• dist[0] = 0

• dist[1] = INF • dist[2] = INF

• dist[3] = INF

Step 2: First Iteration (i=0) - Relax all edges

• Edge 0: dist[0] + 10 = 10 < INF → dist[1] = 10

• Edge 1: dist[1] + 20 = 10 + 20 = 30 < INF → dist[2] = 30

• Edge 2: dist[2] + 30 = 30 + 30 = 60 < INF → dist[3] = 60

• Edge 3: dist[3] + (-60) = 60 + (-60) = 0 ≥ dist[0] → no change

After iteration 1: dist = [0, 10, 30, 60]

Step 3: Second Iteration (i=1) - Relax all edges

• Edge 0: dist[0] + 10 = 10 ≥ dist[1] → no change

• Edge 1: dist[1] + 20 = 10 + 20 = 30 ≥ dist[2] → no change

• Edge 2: dist[2] + 30 = 30 + 30 = 60 ≥ dist[3] → no change

• Edge 3: dist[3] + (-60) = 60 + (-60) = 0 ≥ dist[0] → no change

After iteration 2: dist = [0, 10, 30, 60] (no changes)

Step 4: Third Iteration (i=2) - Relax all edges

• All edges: no changes (same calculations as iteration 2)

After iteration 3: dist = [0, 10, 30, 60] (no changes)

Step 5: Fourth Iteration (i=3) - Check for negative cycles

• Edge 0: 0 + 10 = 10 ≥ 10 → no change

• Edge 1: 10 + 20 = 30 ≥ 30 → no change

• Edge 2: 30 + 30 = 60 ≥ 60 → no change

• Edge 3: 60 + (-60) = 0 ≥ 0 → no change

Step 6: Analysis

• No distance updates occurred in the 4th iteration

• Therefore, no negative cycle reachable from node 0 exists

Output:

```
not possible
```

Pseudocode: INPUT testCount

```
FOR t = 1 TO testCount:
    INPUT numNodes, numEdges
    edgesList = []
    FOR i = 1 TO numEdges:
        INPUT fromNode, toNode, timeShift
        ADD (fromNode, toNode, timeShift) TO edgesList


    earliestTime = [INFINITY] * numNodes

    earliestTime[0] = 0


    FOR i = 1 TO numNodes:

        FOR each (start, end, weight) IN edgesList:

            IF earliestTime[start] != INFINITY AND earliestTime[start] + weight < earliestTime[end]:

                earliestTime[end] = earliestTime[start] + weight


    hasNegativeCycle = false

    FOR each (start, end, weight) IN edgesList:

        IF earliestTime[start] != INFINITY AND earliestTime[start] + weight < earliestTime[end]:

            hasNegativeCycle = true


    PRINT "possible" IF hasNegativeCycle ELSE "not possible"
```

Code: https://github.com/pulokdas062/Graph-Algorithm/edit/main/Bellmanford/UVA558/UVA558.cpp